Design of a High-Throughput Match Search Unit for Lossless Compression Algorithms

Anonymous 1
ACME inc.
Omitted due to blind review

Anonymous 2
ACME inc.
Omitted due to blind review

Anonymous 3
ACME inc.
Omitted due to blind review

Anonymous 4
ACME inc.
Omitted due to blind review

***Abstract*—This paper presents an attempt to combine recent research in fields of hardware- and software-based high-throughput universal lossless compression algorithms and their implementations, resulting into a case study focusing on one of the most critical parts of compression algorithms – a Match Search Unit (MSU). The presented FPGA design combines ideas of the LZ4 algorithm (which is derived from the most common LZ77) with the state of the art hardware architectures for lossless compression also based on LZ77. This approach might lead to a smaller, better organized or more efficient "building block" for modern implementations of hardware driven lossless compression algorithms. The presented design focuses on optimization of the main problem of the LZ77 family, namely the construction of and searching in a compression dictionary. Particularly, we combine a Live Value Table (LVT) with multi-ported memory in order to improve the bandwidth of the dictionary and the Fibonacci hashing principle originating from LZ4 algorithm to decrease latency of the MSU and to achieve overall higher throughput rate. For the design synthesis an FPGA of the Xilinx Virtex-7 family was used.**

***Index Terms*—FPGA; high; bandwidth; fast; lossless; compression; algorithm; architecture; LZ4; LZ77; hash; table; LVT; multiport; memory; Xilinx; Virtex**

I. Introduction

In recent decades throughput of lossless compression systems has increased by an order of magnitude [1] . Further progress requires new and more complex architectures. Adapting certain techniques from software domain into hardware domain can generally increase performance or decrease consumption of FPGA logic resources opening the possibility of accelerators with higher density and frequency on a single chip. This approach is crucial in overcoming the challenge of 100 Gbps throughput which is expected to be needed in upcoming years. However, designing a block with approximately 10 Gbps throughput would be a good testing setup for implementation of optimizations from software domain. We would like to scale-up our solution towards 100 Gbps in the near future.

The rest of the paper is structured as follows. In Section II we summarize relevant previous works and we analyze key features of lossless compression algorithms, their software implementations and current FPGA architectures, which may have influence on compression performance. In Section III we propose an optimized FPGA design of a Match Search Unit. Performance measurements of this design are presented in Section IV. Finally, we propose further possible improvements in Section V.

II. State of the Art & Analysis

The era of an universal lossless compression has begun in 1977 with the introduction of the first Lempel-Ziv algorithm known as the LZ77 [2]. Many following algorithms have been based upon combination of LZ77 and Huffman Encoding principle [3]. These methods resulted in DEFLATE algorithm [4], which together with bare LZ77 represents the most widespread implementations of lossless compression among hardware architectures.

Few years ago, a concept of "fast" compression algorithms [5], trading the compression ratio for speed, has appeared. Representative examples of "fast" algorithms are LZO [5] or LZ4 [6], which are using advantage of LZ77's single-pass (de-)compression, where the input data are processed only once. Thanks to this feature, the hardware and software implementations are highly efficient and compact, and commonly benefit from hardware-tight optimizations, such as pipelining, loop unrolling or cache optimization [5].

A. LZ77 FPGA implementations and the effect of an MSU

Several architectures of an LZ77-based compression scheme in FPGA have been recently presented [8]–[11]. These implementations focused on real-time lossless compression of IP (Internet Protocol) packets at 10 Gbps throughput (or 40 Gbps in the most recent case [8]).

General architecture of LZ77-based implementation consists of two parts. The first is a "Match Search Unit" (MSU) which is used to find indexes of repeated occurrences throughout the processed data. The second is an encoding unit which transforms the occurrences into given output format. This part is usually a simple finite state machine which has very little effect on compression ratio and speed of the design. The MSU is the core element in every LZ77-based design. It is usually the most complex part of a compression algorithm with highest impact on its speed. It contains a data structure (compression dictionary), which allows it to store and index a certain amount of history of processed data. This allows the MSU to quickly find a match between the already processed data and the data being processed. This dictionary therefore affects the overall speed of the MSU. The performance of the MSU can be affected by the following properties of the dictionary:

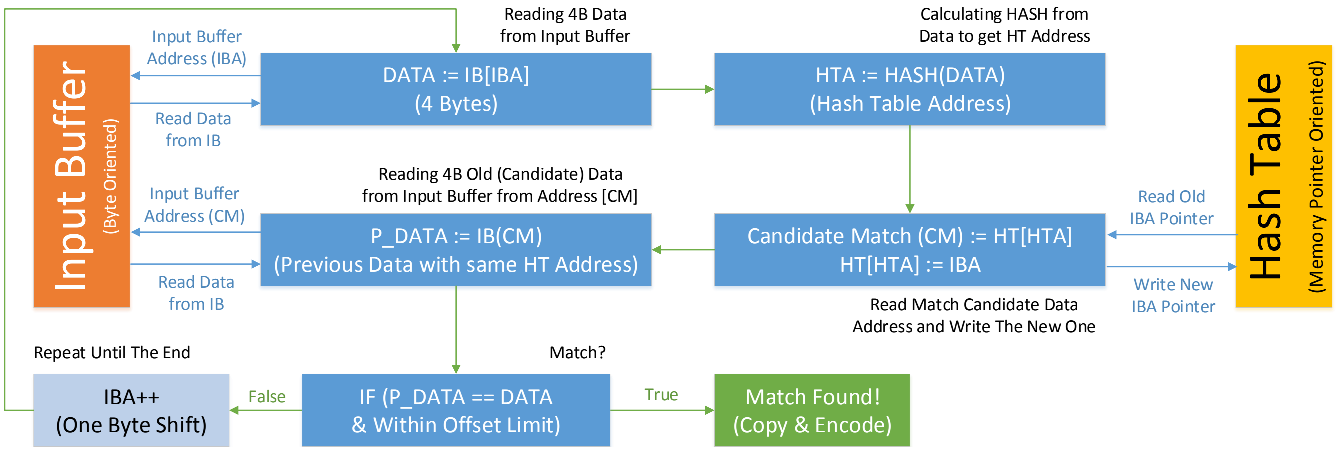

Input Buffer
(Byte Oriented)
Input Buffer Address (IBA)
Read Data from IB
Input Buffer Address (CM)
Read Data from IB
Reading 4B Data from Input Buffer
DATA := IB[IBA] (4 Bytes)
Calculating HASH from Data to get HT Address
HTA := HASH(DATA) (Hash Table Address)
Hash Table
(Memory Pointer Oriented)
Reading 4B Old (Candidate) Data from Input Buffer from Address [CM]
P_DATA := IB(CM) (Previous Data with same HT Address)
Candidate Match (CM) := HT[HTA] HT[HTA] := IBA
Read Old IBA Pointer
Write New IBA Pointer
Read Match Candidate Data Address and Write The New One
Repeat Until The End
Match?
IBA++ (One Byte Shift)
False
IF (P_DATA == DATA & Within Offset Limit)
True
Match Found! (Copy & Encode)

Fig. 1. Data flow of the original LZ4 MSU [6], [7].

1) The compression ratio is directly proportional to the size of the dictionary. However, a larger dictionary uses more FPGA resources and in most cases it reduces the design speed and therefore the compression speed is being reduced as well.
2) The number of independent ports enabling parallel search for matches can increase the throughput.
3) The overall architecture of the dictionary – sub-dictionaries [9] vs. fully shared [8] vs. other techniques.

Current trend in hardware implementations of dictionary structures is to use a hash table [6]–[9], rather than a Content Access Memory (CAM). However, both of them are more suitable for hardware domain, instead of more sophisticated data structures such as binary trees, which are commonly used in software implementation with higher compression ratios. Due to the fact that the duration of the data access is not fixed, these structures are usually not suitable for hardware implementations.

B. *Hash Tables & Hash Algorithms Properties*

In contrast to CAM, where each data input is stored in ağiven memory slot, hash table uses hashing principle [12] to create hash (key) for certain input. This input is then used as an address for assigning a memory slot to the input data. The hashing principle is an essential technique for efficient implementation of any data storage system realized in hardware or software.

The key can be adjusted according to the size of the memory, this allows us to use smaller memory in implementations. However, this causes a problem, when multiple keys are adjusted to the same memory slot (collision). The number of collisions belongs to performance indicators of the hashing principle, most of the hashing principles with good performance have key distribution almost uniform.

In software domain the collisions are solved by hashing the data again (rehashing). This approach can be used in hardware designs [7], however, in high throughput cases it is not a very usable approach due to its inconsistent duration, which causes that the design is unable to guarantee minimal performance. Therefore in high throughput compression designs collisions are ignored to guarantee the minimal throughput performance, this results in negative effect on compression ratio but it usually shows an acceptable degree.

In hardware design the size of memory is usually given in a form of 2^n. Some of the hashing principles are using prime numbers to achieve best performance, one of them being the Fibonacci hashing principle [12]. This principle is based upon simple multiplication with constant which is the closest possible prime number to the golden ratio.

C. *LZ4 Algorithm*

LZ4 was chosen as a candidate for our design following a full analysis of LZ4 [6] from a hardware designer's perspective. The advantageous chracteristics of LZ4 include the highest (de-)compression speed among other lossless algorithms, resource efficiency and easy implementation in an FPGA. The current highest throughput implementation of LZ4 algorithm can reach up to 4 Gbps [13].

The data flow of the LZ77 compression family is divided into three parts – the compression stream, output stream and buffer for the decompressed data. The main principle of this family is that the compression stream is divided into segments called ”data blocks”. Each block consists of literals copied directly to the output stream, and back reference used to copy certain number of bytes from the already decompressed buffer to the output stream. This technique is called de-duplication, which represents an origin of the lossless compression in the LZ77 family.

As described in [6], [7], a software variant of the LZ4 compression algorithm is divided into four parts:

1) input buffer,
2) Match Search Unit (MSU) with a hash table,
3) output sequence encoder when a match occurs,
4) output buffer.

The data flow of the LZ4 MSU [6], [7] is depicted in Fig. 1. The picture also indicates a potential problem for parallelization, namely the write operations being performed over the same memory segment of the *Hash Table*. These operations are spread over the whole memory space, therefore they cannot be merged into a single transaction. We experience a similar issue also with read operations which are random as well. These two problems can be solved by the use of a multiport memory for the Hash Table (with multiple readwrite ports) and an input buffer (multiple read ports). The number of read and write ports should be equal to the degree of the planned parallelism.

D. Multiport Memory in FPGAs

A multiport memory is a component essential for implementation of a ”fully-shared” dictionary, where the content is shared among all MSUs. The Live Value Table (LVT) [14] is the most common approach for creating a multiport memory from an ordinary single port (or dual port) memory, such as Xilinx BRAM or Altera M9K blocks.

There is also an alternative approach using an XOR technique [22], [23] for accessing the latest value. The main advantage of this method is reducing the number of logic elements and increasing speed for low-depth memories. However, for larger memories this method has lower frequency which is an essential parameter for high throughput design.

Before the introduction of the LVT, multiport memory was implemented usually from general re-configurable logic. Such logic has in theory no limits in capacity and parallelism (the number of ports), but it requires a lot of FPGA resources, thus slowing the whole design down in terms of its operating frequency and requiring a large area of an FPGA for implementation.

The idea of the LVT is to divide such a memory design using general re-configurable logic into two parts:

1) Data memory using single or dual port embedded FPGA memory blocks (BRAMs) where the number of read-/write ports can be increased by replication, banking and multipumping principles [14].
2) Control memory (created from general logic) keeps the information for each written address concerning in which part of the data memory the latest value is stored. With each read operation, this information is used to set output multiplexers to the location of the latest value written for the given address.

III. Our Approach

In this section we describe decisions we made within the architecture design. The initial idea was to combine features mentioned in the previous section into a new architecture of the MSU. We started with the original LZ4 architecture which is depicted in Fig. 1.

A. Implementation Platform & Minimum Throughput

We selected the Xilinx 7-Series logic, which has been used for the latest three FPGA generations and appears to be a good practical choice, however the described architecture can be easily implemented in any modern FPGA. The recent architectures [8], [9] achieved 10 Gbps throughput. A common 10G Ethernet IP core uses a 64-bit data path clocked at 156.25 MHz [15]. We aim our design to be used currently for 10G applications with future development towards 40G and 100G Ethernet applications.

B. Speed Optimization of the LZ4 Original MSU

We investigated a way to improve the performance of the original LZ4 MSU hardware design Fig. 1. This design uses a simple and non-parallel flow of the LZ4 MSU unit. There are several issues in the design:

- For each 4-byte block of memory that is tested for a match, we need to perform two reads from the input buffer and both a read and write to the hash table.
- These accesses to the memories are distributed across the whole memory space, therefore they cannot be merged into a single transaction operation. This causes a problem with the parallelization of the unit, because it requires a multi-ported memory for the input buffer and the hash table (see Fig. 1).
- There is a memory access for a 4-byte block in each cycle, however the memory block of the next cycle has only a single byte offset from the previous cycle (see Fig. 2). This results in redundant operations leading to read efficiency of 25%.

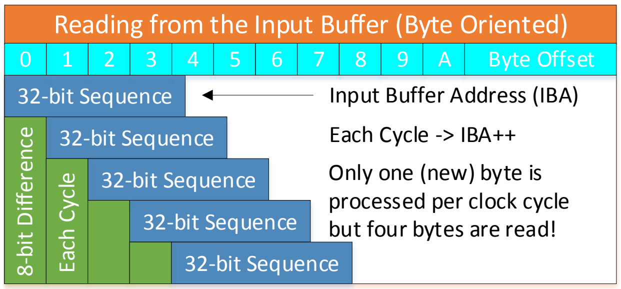

Reading from the Input Buffer (Byte Oriented)
0
1
2
3
4
5
6
7
8
9
A
Byte Offset
32-bit Sequence
Input Buffer Address (IBA)
Each Cycle -> IBA++
Only one (new) byte is processed per clock cycle but four bytes are read!
8-bit Difference
Each Cycle

Fig. 2. Original LZ4 memory read scheme.

There are two fundamental ways of increasing design throughput - running the design at a higher operating frequency or increasing the number of data bits processed per a clock cycle. In our case, increasing the number of data bits is more suitable because compression algorithm implementations tend to be quite complex and a critical path in the logic limits the operating frequency.

1) Increasing Design Datapath Width: We assume that the data for processing are IP packets coming from a 10G Ethernet interface, delivered via a 64-bit data stream clocked at 156.25 MHz. The IP packet payload can be up to 9 kB large (Jumbo packets), to utilize the maximum throughput of the 10G Ethernet. We can use native width of the datapath and size of the payload for our advantage to construct the *Input Buffer* with capacity of 16 kB, 64-bit write port and 128-bit read port (the nearest power of two needed to read the required volume of the input data – 88-bits, see Fig. 3) from FPGA BRAM memory without any significant overhead.

We need to process 64-bits (8 bytes) per a clock cycle to match the requirements of a 10G Ethernet datapath. The original LZ4 MSU can process up to one byte per a clock cycle, but four bytes had to be read from the *Input Buffer* (see Fig. 1). We can make use of the fact that the data in the *Input Buffer* are consecutive and the processed 4-byte data blocks are overlapping with each other, thus we will use 88-bits (8

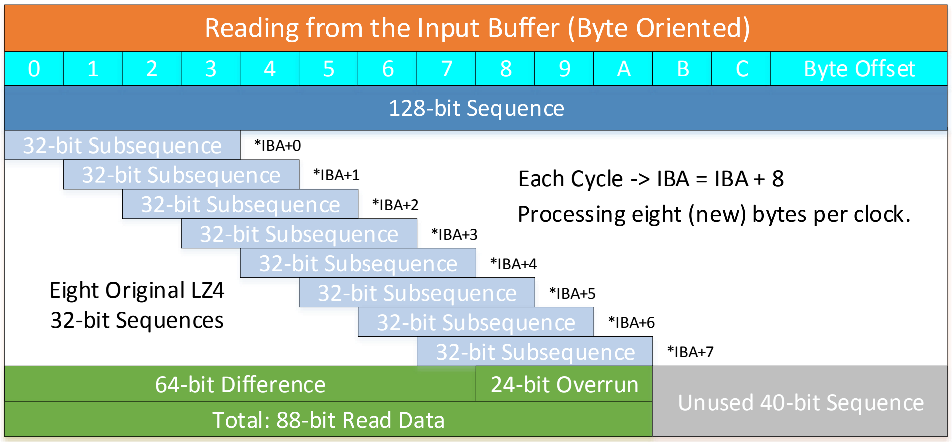


Fig. 3. Hardware optimized LZ4 memory read schema

blocks of 4-byte sequences with 1-byte overlap, last 3-bytes are for the last read) of the 128-bit read interface, which is the nearest higher power of 2. The optimized memory scheme is depicted in Fig. 3.

We need to generate eight pairs of *Input Buffer Address (IBA)* pointers where each destination represents one of the 4-byte blocks. Address pointers are also consecutive, therefore they can be calculated by a simple adder logic. Thanks to that only the "master" *IBA* pointer is needed, which will be increased by 8 (originally by 1) in every clock cycle until a match is found. Finally, this approach does not increase the number of access operations to the *Input Buffer*, but it increases throughput eight times and the overall memory read efficiency to the range of 50%–72% (considering a 128-bit raw sequence or 88-bit read data, respectively) compared to the original 25%.

2) Reducing Number of Read/Write Memory Accesses: The major issue is the parallel access to the *HT* (Hash Table) representing a dictionary in this particular case and the following access operations to *IB* (readback of candidate match positions). Data are stored in the *HT* at nearly random positions (they are no longer consecutive as the data before the hashing phase). We decided to implement a "fully shared" dictionary (unlike 842B [9] technique or any multibank-based principle), where all MSUs have access to a single dictionary (behaving as a single memory). Without this dictionary the compression ratio would be significantly decreased because matching would occur only between blocks processed by the same *IBA* pointer. This decision requires to implement a multiport memory using the LVT [14] technique. Therefore all read/write operations in *HT* are accessing random addresses.

The original algorithm reads an old *IBA* after the hash calculation and stores a new one in its place. The architecture of the FPGA embedded block memory allows performance of both operations as a single transaction in a single clock cycle. The *Hash Table Address (HTA)* is obtained via a hash function from data read from *IB*.

Our idea is to reduce the number of accesses via merging the processed data and the IBA pointer into a single record inside the Hash Table (see Fig. 4). This approach will remove the need for the following read accesses from *IB*. The *HT* becomes the only component requiring a multiport memory. The resulting architecture will be simplified because only one LVT based memory will be required.

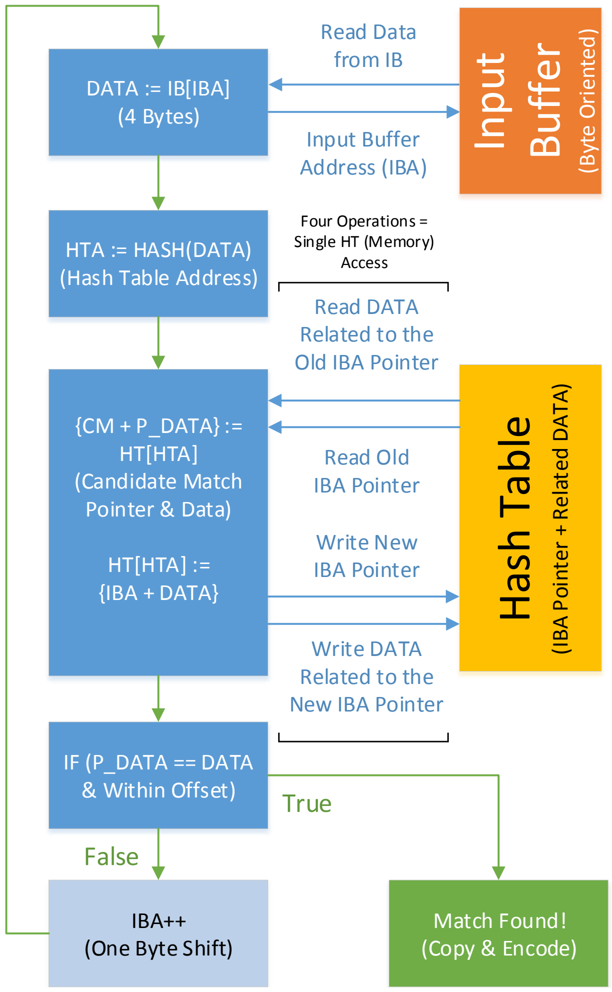
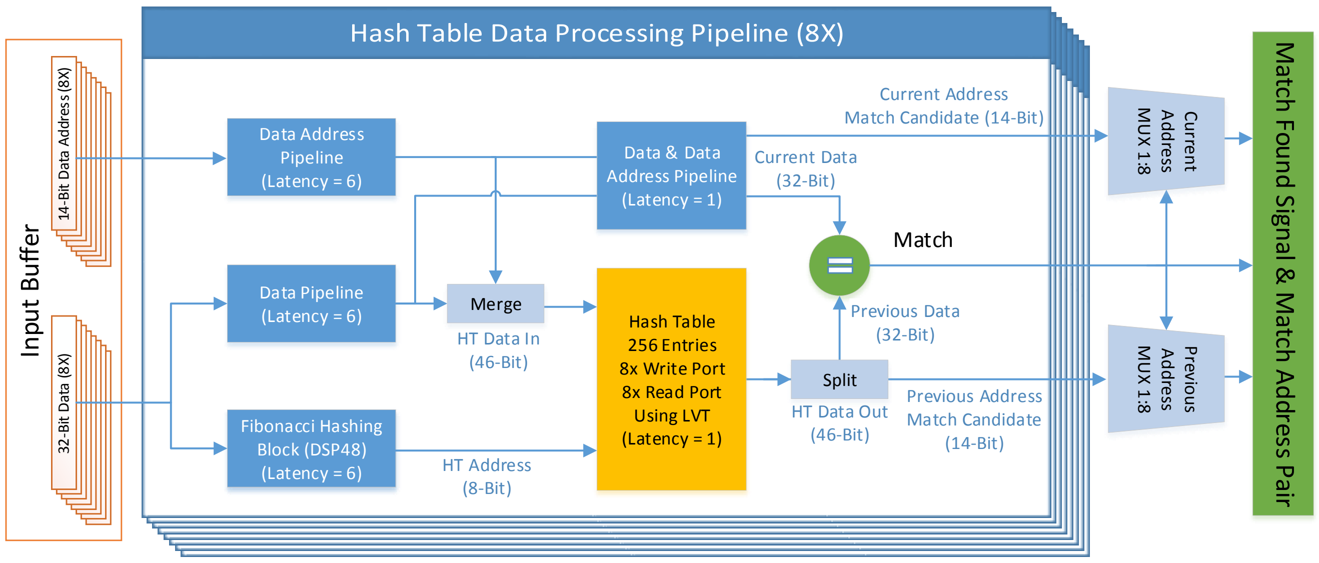


Fig. 4. Flow of the (memory access) optimized LZ4 MSU

3) Hash Calculation & Pipelining: The last important part required for the MSU architecture is the hash calculation block. The original LZ4 uses a Fibonacci hashing principle where the input value is multiplied by a constant (a prime



Hash Table Data Processing Pipeline (8X)
Input Buffer
14-Bit Data Address (8X)
32-Bit Data (8X)
Data Address Pipeline (Latency = 6)
Data Pipeline (Latency = 6)
Fibonacci Hashing Block (DSP48) (Latency = 6)
Merge
HT Data In (46-Bit)
HT Address (8-Bit)
Data & Data Address Pipeline (Latency = 1)
Hash Table 256 Entries 8x Write Port 8x Read Port Using LVT (Latency = 1)
Current Address Match Candidate (14-Bit)
Current Data (32-Bit)
Match
Previous Data (32-Bit)
Split
HT Data Out (46-Bit)
Previous Address Match Candidate (14-Bit)
Current Address MUX 1:8
Previous Address MUX 1:8
Match Found Signal & Match Address Pair

Fig. 5. Architecture of the new MSU inspired by the (hardware) optimized LZ4 flow

number) and higher bits are selected to create the *Hash Table Address (HTA)* [6].

The Xilinx DSP48 block can be used for the hash calculation. The optimal settings (recommended by the Xilinx CoreGen tool) for multiplication of two 32-bit numbers are: four DSP48 blocks and the calculation requires six clock cycles (estimation made the CoreGen) thus the block will be able to operate at approximately 700 MHz in case of a Virtex-7 chip [16]. The pipelining principle is required to mask the computation latency and to maximize the throughput. We can also utilize maximum operating frequency by using higher frequency for the DSP48 blocks than for the rest of the design, thus reducing the length of the pipeline for hash calculation to 3 or 2 cycles.

IV. Results

In this section we discuss the contribution, designed architecture, measurement setup and results comparison.

A. Optimized Architecture

Throughput of at least 10 Gbps requires an 8-way parallel architecture, which is the minimum configuration for the original byte-oriented LZ4 flow. The architecture requires an 8-port hash table implementing the dictionary (for 256-1024 entries), thus the *HT* memory will use the LVT principle. The *IB* (Input Buffer) has the capacity of 16 kB (for Jumbo packet support) and has been optimized to process an aligned 128-bit read operation instead of original 32-bits. The LVT principle is not required for the *Input Buffer*. The architecture (depicted in Fig. 5) is based on the optimized LZ4 flow, which is more suitable for implementation in an FPGA then the original LZ4 flow 1. The architecture is capable of processing eight bytes in a single clock cycle.

The FPGA design expects 8 pairs of *IBA* pointers and the related data (32-bit long sequences), both provided by the *IB*. These pairs are pipelined along a hash calculation block, where 8 *HTA*s are calculated. The length of the pipeline must be the same as the latency of the hash calculation blocks which are clocked at a higher frequency to decrease the latency. These pairs will be read and written to the locations specified by *HTA*s in the *HT* with the latency of one clock cycle. Each pair is also pipelined along the *HT* representing the compression algorithm dictionary.

The previous data (*P_DATA*) are extracted from a candidate match (*CM*) pair, which is read from the *HT*. If the previous data are equal to the current data which is read from the pipeline, a match occurs. Both previous and current IBAs will become a match. The last step is to resolve possible multiple matches in a single clock to a single match via multiplexers controlled by a priority encoder (the lower the value of *IBA*, the higher the priority).

B. Measurement Setup & Synthesis Results

The Xilinx ISE 14.7 toolkit was used to synthesize the presented architecture for the Xilinx Virtex-7 XC7V330T-2FFG1157 chip. For the synthesis the optimizations were adjusted to *Speed/High* . The hash table size was set to the value of 256 entries (minimal feasible value [21]). The FPGA resource utilization for the routed design is summarized in Table I. We used the random FPGA pin placement feature available in the Xilinx ISE toolkit to get a fully routed design. This feature allowed us to measure the design speed in a more realistic way.

TABLE I
LZ4 MSU resources utilization for Virtex-7 XC7V330T

Slice	LUT	Flip Flop	BRAM	DSP48	Frequency
4828	15014	8530	64	32	250 MHz

C. Resource Use Comparison

We know that the Xilinx 7-Series logic *SLICE* block contains four LUT6 (6-input Look Up Table) blocks plus eight flip-flops [17]. We also know that the Altera Stratix V ALM contains two LUT6 and 4 flip-flops [19], thus a single Xilinx Slice can be considered equal to two Altera ALMs.

This simplification allows us to compare our architecture to the previous work [8]. The *PWS=8 w/ HT* (Parallelization Window Size) variant can process 8 bytes per a clock cycle, therefore we have selected this variant for a brief comparison (see Tab. II). Our architecture excludes the input and output buffers and the output sequence encoding part. However, the amount of FPGA logic resources are comparable (PWS=8 w/ HT variant has 16519 ALMs compared to 9656 ALMs) from perspective of the used FPGAs. The overall latency of the [8] is 41 clock cycles. The parts of the architecture [8] which are implemented also in our approach include: a hash calculation, a hash table update, a string match and a match selection. The latency of these selected parts was originally 29 clock cycles, whereas our approach has the latency of 7 clock cycles only, thus the latency has been decreased approximately four times.

TABLE II
BRIEF MSU PERFORMANCE COMPARISON

Solution	ALMs	Latency [Cycles]	Throughput [Gbps]
Our LZ4 MSU	9656	7	16,0
PWS=8 w/HT [8]	16519	29 (41)	11,2
LZ4 ASIC [13]	N/A	17	4,0
LZ4 8-Bit [7]	690	N/A	2,0

1) Compression Ratio Analysis and Simulation: However, it is difficult to compare compression ratios of this approach against other implementations because our MSU architecture is only part of compression architecture. This is the reason, why we developed a functionally equivalent software model of our hardware architecture. This software model was connected to an existing software LZ4 architecture re-using other parts of LZ4 (input buffer, output buffer and the LZ4 encoding algorithm). We assume the compression ratio of the software model has to be equal to the presented hardware architecture, because the process of output encoding cannot affect the compression ratio (see Fig. 6).

We processed three compression corpuses (Calgary [24], Canterbury [25], Silesia [26]) through our (hardware based) software model of LZ4 to obtain experimental results. The compression ratio is scaling up in an expected way in relation to the size of the hash table (see Tab. III).

TABLE III
COMPRESSION RATIOS FOR DIFFERENT HT SIZES

HT Size	Calgary	Canterbury	Silesia
64	1,28	1,40	1,24
128	1,34	1,46	1,28
***256**	1,38	1,5	1,31
512	1,39	1,57	1,32
1024	1,47	1,68	1,4
2048	1,54	1,75	1,48
4096	1,61	1,81	1,56

*The size of 256 entries is used for the presented MSU architecture.

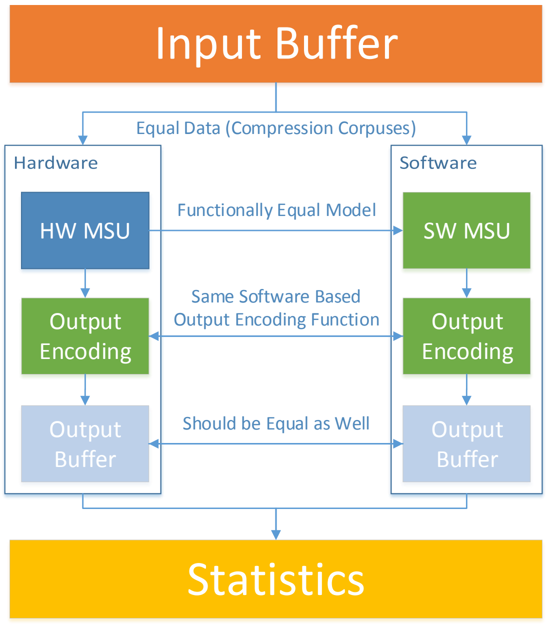


Fig. 6. Experimental simulation flow for obtaing compression ratios.

Our compression ratio is lower than LZ4 8-bit [7] and Software LZ4 due to their usage of advanced collision handling. We assume that the LZ4 8-bit architecture uses 24-bit wide address of the input buffer, thus 32 or 64 BRAMs are used for implementation of the dictionary (the most feasible configurations). All remaining BRAMs realize the input and the output buffer. The worst case of 32 BRAMs is representing the dictionary size of 65536 entries [18] compared to our 256 entries.

Both designs are using rehashing principle, which enables them to achieve higher compress ratio, however they cannot guarantee their worst case throughput. LZ4 8-bit [7] is only providing peak performance throughput, which is derived from processing 8 bytes per clock. We are presenting MSU with worst case performance, which is an essential characteristics of stable high-throughput design.

TABLE IV
BRIEF COMPRESSION RATIO COMPARISON

Solution	Calgary	Canterbury	Silesia
Our LZ4 MSU	1,38	1,5	1,31
PWS=8 w/HT [8]	1,82	N/A	N/A
LZ4 ASIC [13]	N/A	N/A	N/A
LZ4 8-Bit [7]	Incomplete (1,65 – 2,05)		
Ref SW LZ4	2,26	2,11	2,41

2) Influence of the Latency: An MSU (and overall system) latency affects the system readiness to accept new data. Lower latency also allows us to ”squeeze” more data into constant throughput media by lowering an inter-frame gap, in a packet oriented real-time systems (with packets processed one by one, not in a continuously streamed manner), in case that more parallel blocks are used at the same time.

All MSU architectures designed for high throughput applications use pipelining principle where data are processed alongside control signal or other data (for example hash calculations or matching) in several stages. The latency reduction might enable the number of pipeline stages to decrease resulting in lower usage of logic resources.

V. Future Work

We are considering to design new optimizations for the presented architecture. The first idea is to design an optimized hash calculation unit which by itself is currently capable of running at approximately 700 MHz [20] with the latency of 6 clock cycles, whereas the current design can run up to 250 MHz (and cannot be further significantly increased). The architecture of a DSP48 block allows to bypass some pipelining stages to limit the latency thus limiting the frequency. We can reduce the latency to just 3 clock cycles while matching the frequency of DSP48 blocks to the rest of the MSU design. Therefore we can save additional resources in *Data* and *Data Address* pipelines (see Fig. 5). The reduction in this particular case will be 50% of flip-flops required for these pipelines.

We can also apply similar principle to an embedded memory block (also capable of running at approx. 600 MHz [20]) to implement the multipumping principle [14] in order to save resources (BRAMs) required for the LVT based hash table. However the multipumping principle will increase the amount of other used FPGA resources. Further optimizations of the LVT principle will be also explored [21].

We started implementating a system realizing the LZ4 lossless compression algorithm with use of the presented MSU architecture with the aim of minimum 10 Gbps throughput.

VI. Conclusion

We presented an architecture of a Match Search Unit (MSU) inspired by a modern fast LZ4 lossless compression algorithm suitable for hardware implementations. We proposed optimizations of the original LZ4 flow for reducing the memory read/write accesses towards the implementation platform (Xilinx Virtex-7 FPGA logic).

The latency of the presented MSU solution has been reduced 4 times compared to the previous work [8], while the amount of FPGA logic resources is comparable.In contrast to [7] our MSU architecture guarantees minimal throughput, has substantially higher throughput rate with comparable amount of FPGA resources and achieves slightly lower compression ratio with use of significantly smaller dictionary size (just 256 entries respectively 65536).

The design has the potential for further optimizations in order to significantly reduce the overall latency and resource consumption. For the current implementation the theoretical throughput is 16 Gbps.

Acknowledgment

Omitted due to blind review.
Omitted due to blind review.
Omitted due to blind review.
Omitted due to blind review.
Omitted due to blind review.
Omitted due to blind review.

References

[1] Cisco Visual Networking Index: Forecast and Methodology, 2008–2013, [Online] https://tinyurl.com/yb6wf4k3

[2] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," in IEEE Transactions on Information Theory, vol. 23, no. 3, pp. 337-343, May 1977. doi: 10.1109/TIT.1977.1055714

[3] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," in Proceedings of the IRE, vol. 40, no. 9, pp. 1098-1101, Sept. 1952. doi: 10.1109/JRPROC.1952.273898

[4] D. Harnik, E. Khaitzin, D. Sotnikov and S. Taharlev, "A Fast Implementation of Deflate," 2014 Data Compression Conference, Snowbird, UT, 2014, pp. 223-232. doi: 10.1109/DCC.2014.66

[5] J. Kane and Q. Yang, "Compression Speed Enhancements to LZO for Multi-core Systems," 2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing, New York, NY, 2012, pp. 108-115. doi: 10.1109/SBAC-PAD.2012.29

[6] M. Bartík, S. Ubik and P. Kubalík, "LZ4 compression algorithm on FPGA," 2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS), Cairo, 2015, pp. 179-182. doi: 10.1109/ICECS.2015.7440278

[7] W. Liu, F. Mei, C. Wang, M. O'Neill and E. E. Swartzlander, "Data Compression Device Based on Modified LZ4 Algorithm," in IEEE Transactions on Consumer Electronics, vol. 64, no. 1, pp. 110-117, Feb. 2018. doi: 10.1109/TCE.2018.2810480

[8] J. Fowers, J. Y. Kim, D. Burger and S. Hauck, "A Scalable High-Bandwidth Architecture for Lossless Compression on FPGAs," 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, Vancouver, BC, 2015, pp. 52-59. doi: 10.1109/FCCM.2015.46

[9] B. Sukhwani, B. Abali, B. Brezzo and S. Asaad, "High-Throughput, Lossless Data Compresion on FPGAs," 2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines, Salt Lake City, UT, 2011, pp. 113-116. doi: 10.1109/FCCM.2011.56

[10] R. Mehboob, S. A. Khan, Z. Ahmed, H. Jamal and M. Shahbaz, "Multigig lossless data compression device," in IEEE Transactions on Consumer Electronics, vol. 56, no. 3, pp. 1927-1932, Aug. 2010. doi: 10.1109/TCE.2010.5606348

[11] K. Papadopoulos and I. Papaefstathiou, "Titan-R: A Reconfigurable Hardware Implementation of a High-Speed Compressor," 2008 16th International Symposium on Field-Programmable Custom Computing Machines, Palo Alto, CA, 2008, pp. 216-225. doi: 10.1109/FCCM.2008.14

[12] E. D. Knuth, "The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching", 1998, ISBN 0-201-89685-0. Addison Wesley Longman Publishing Co., Inc.

[13] Sang Muk Lee, Ji Hoon Jang, Jung Hwan Oh, Ji Kwang Kim, Seung Eun Lee, Design of hardware accelerator for Lempel-Ziv 4 (LZ4) compression, IEICE Electronics Express, ISSN 1349-2543 , doi: 10.1587/elex.14.20170399

[14] C. LaForest, S. Gregory, "Efficient Multi-ported Memories for FPGAs", Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, CA, 2011, pp. 41-50. doi: 10.1145/1723112.1723122

[15] Z. Bradac, S. Valach, 10G bit ethernet phy implementation in FPGA based systems, In IFAC Proceedings Volumes, Volume 39, Issue 21, 2006, pp. 427-432, ISSN 1474-6670. doi: 10.1016/S1474-6670(17)30224-0.

[16] 7 Series DSP48E1 Slice User Guide (UG479), Xilinx [Online]. Available: https://tinyurl.com/ybhx4r93

[17] 7 Series FPGAs Configurable Logic Block User Guide (UG474), Xilinx [Online]. Available: https://tinyurl.com/xug474

[18] 7 Series FPGAs Memory Resources (UG473), Xilinx [Online]. Available: https://tinyurl.com/y75gctmw

[19] Stratix V Device Handbook (SV-5V1 2017.12.15), Altera [Online]. Available: https://tinyurl.com/stx5-alm

[20] Virtex-7 T and XT FPGAs Data Sheet – DC and AC Switching Characteristics (DS183), Xilinx [Online]. Available: https://tinyurl.com/v7-ds183

[21] M. Bartík, S. Ubik and P. Kubalík, ”A novel and efficient method to initialize FPGA embedded memory content in asymptotically constant time,” 2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, 2016, pp. 1-6. doi: 10.1109/ReConFig.2016.7857146

[22] Charles Eric Laforest, Zimo Li, Tristan O'Rourke, Ming G. Liu, and J. Gregory Steffan. Composing multi-ported memories on fpgas. ACM Trans. Reconfigurable Technol. Syst., 7(3):16:1–16:23, September 2014.

[23] Ameer M.S. Abdelhadi and Guy G.F. Lemieux. Modular multi-ported sram-based memories. In Proceedings of the 2014 ACM/SIGDA International Symposium on Field-programmable Gate Arrays , FPGA '14, pages 35–44, New York, NY, USA, 2014. ACM.

[24] Bell, T.C., Witten, I.H. and Cleary, J.G. ”Modeling for text compression,” Computing Surveys 21(4): 557-591; December 1989; ISSN: 0360-0300. doi: 10.1145/76894.76896

[25] R. Arnold and T. Bell, ”A corpus for the evaluation of lossless compression algorithms,” Proceedings DCC '97. Data Compression Conference, Snowbird, UT, USA, 1997, pp. 201-210. doi: 10.1109/DCC.1997.582019

[26] Deorowicz, S., Universal lossless data compression algorithms, Silesian University of Technology, 2003;